Are Sparse Representations Really Relevant for Image Classification ? *

Roberto Rigamonti, Matthew A. Brown, Vincent Lepetit
CVLab, EPFL
Lausanne, Switzerland
firstname.lastname@epfl.ch

Abstract

Recent years have seen an increasing interest in sparse representations for image classification and object recognition, probably motivated by evidence from the analysis of the primate visual cortex. It is still unclear, however, whether or not sparsity helps classification. In this paper we evaluate its impact on the recognition rate using a shallow modular architecture, adopting both standard filter banks and filter banks learned in an unsupervised way. In our experiments on the CIFAR-10 and on the Caltech-101 datasets, enforcing sparsity constraints actually does not improve recognition performance. This has an important practical impact in image descriptor design, as enforcing these constraints can have a heavy computational cost.

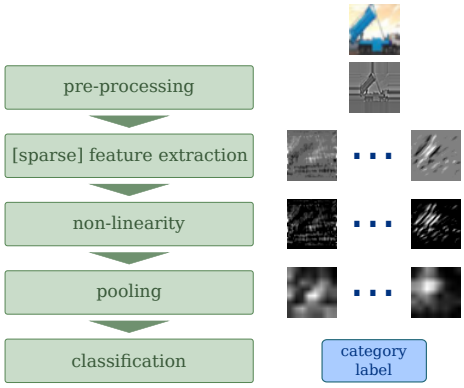


Figure 1: Architectural model of the classification pipeline. On the right side, the evolution of the internal representation is depicted for a sample truck image taken from the CIFAR-10 dataset [14].

1. Introduction

Inspired by work from the neuroscience community [24, 33], there has been increasing interest in computer vision algorithms that rely on sparse image representations [28, 23, 35, 37, 2, 34]. The effectiveness of sparse coding from a generative point of view, for image restoration for example [23], is justified by observing that natural images represent only a tiny part of the image space.

While sparse representations have been regarded as more likely to be separable in high-dimensional spaces [29], and therefore suitable for classification, it is still not clear if they are actually needed for recognition tasks. To the best of our knowledge, there is no empirical study showing that sparse representations improve recognition performance compared to non-sparse ones.

Understanding if sparse representations are really relevant for image classification is not only interesting from a theoretical point of view, but also from a practical stance. Their computation typically requires solving either an NP-hard problem or an alternative problem that still involves a costly iterative optimisation [6].

In this paper we aim to evaluate the actual importance of sparsity in image classification, by performing an extensive empirical evaluation and adopting the recognition rate as a criterion. Using the biologically-inspired modular architecture represented in Fig. 1 and closely related to the ones used in [21, 30, 12, 4], we assess the relevance that this property has when both handcrafted and learned filter banks are used in the feature extraction stage.

We experimented on the very challenging CIFAR-10 dataset [32, 14], made of 32×32 images of objects belonging to 10 different categories, and on the more commonly used Caltech-101 dataset [7]. The results of our experiments advocate that no advantage is gained by imposing sparsity at run-time, at least when this sparsity is not tailored in a discriminative fashion. The expensive optimization procedure can therefore be replaced by simple convolutions without affecting the recognition rate.

Even though the main point of our study was not getting a high classification rate per se, our best setup found after extensive experiments obtains results that are comparable

*This work has been supported in part by the Swiss National Science Foundation.

to the state-of-the-art on the CIFAR-10 dataset, but using grayscale images in place of the colour ones exploited by the competing approaches [27, 36, 15].

2. Related Work

Imposing sparsity constraints has recently become popular in computer vision, especially for image classification tasks. This is probably due to evidence for sparse representations in the mammal brain, and because simple algorithms based on such constraints can reproduce linear filters similar to receptive fields observed in V1, the first layer of the visual cortex [24, 29]. These approaches are also attractive because they require only unlabeled data, which are much simpler to produce than labeled data.

As a result, such algorithms have been used to extract features that are assumed to be relevant for classification tasks [26, 12, 34]. Sparsity is also convenient to constrain over-complete linear representations.

Sparse coding can be interpreted as learning the input data distribution with a sparse prior [24]. Deep Belief Networks (DBNs) have been used to learn the data distribution and extract features in an unsupervised way [8], and [18] showed that sparsity constraints are useful to make them converge on natural images toward filters that closely resemble their biological counterparts.

Despite their popularity and to the best of our knowledge, sparsity constraints have not really been evaluated in terms of classification performance. Is it really important to adopt a sparse representation to learn how to extract good features? Is it important to enforce sparsity constraints when extracting the features at run-time, even though it can be costly?

Very recently, [12] developed an architecture very close to ours. They showed the importance of taking the absolute value as a non-linear operation between the feature extraction and the pooling stage, and the power of stacking multiple layers. Nonetheless, they did not really evaluate the effects of sparsity, as they only compared learned sparse features against convolutions with random filters.

Zeiler *et al*. [37] introduced a related model that operates in a convolutional sense as our system does, and proposed a new approach that improves the convergence speed by constraining the representation in an annealing fashion. Again, there is no evaluation of the importance of sparsity constraints.

A comprehensive review of the applications of sparsity in the computer vision and pattern recognition domains is presented in [34], but their claim that sparsity is helpful for the classification task is supported only by few experiments in a supervised, or semi-supervised, context, and not in an unsupervised setting.

Predictive Sparse Decomposition [13] is an endeavor to avoid the sparsity optimization by learning a regressor that approximates the optimal reconstruction. The solution provided by the regressor is, however, only an approximation of the true, sparse solution.

This paper represents an attempt to fill the gap represented by the absence of a thorough evaluation of the real contribution of sparse representations in the image classification task, in order to focus future analyses towards the most relevant aspects of image classification architectures.

3. Evaluation Framework

Our framework for evaluation relies on the architecture shown in Figure 1, which is very similar to the ones used in recent works [21, 30, 4], particularly to [12]. We first extract features by using filters which are either learned or handcrafted. These features either result from a simple convolution between the image and the filters, or from a sparse optimization procedure. We apply a non-linear operation to the output, and then we "pool" the features to obtain some robustness to small translations and deformations using different pooling schemes.

We detail below these different parts of our framework, while the results of the evaluation are given in the next section.

3.1. Feature Extraction and Refinement Stage

3.1.1 Learning the filters

To learn image filters, we chose to adopt Olshausen and Field's algorithm [24] (*OLS*), used in many recent works [22, 12, 34] and known to converge well on natural image patches. We have only slightly modified it for more efficiency when processing images.

In [24], Olshausen and Field suggested that V1, the first layer of the visual cortex, builds a sparse representation of the images. Under this assumption and the hypothesis that a perfect reconstruction is attainable, the problem one would like to solve can be stated as

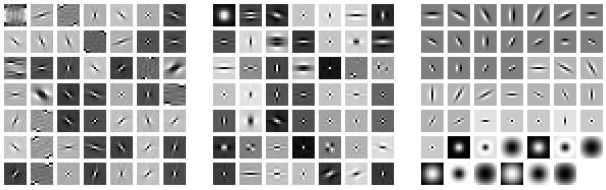
$$\min_{\mathbf{M},\{\mathbf{t}_i\}} \sum_i \|\mathbf{t}_i\|_0 \quad \text{s.t.} \sum_i \|\mathbf{x}_i - \mathbf{M}\mathbf{t}_i\|_2^2 = 0\,, \tag{1}$$

where $\mathbf{x}_i$ are training images, $\mathbf{t}_i$ are the corresponding feature vectors, $\mathbf{M}$ is a matrix whose columns form the dictionary, and the ℓ_0-norm, the number of non-zero elements, is the best sparsity measure available. The ℓ_0-norm formulation in Eq. (1) is, however, non-convex, making the optimization very difficult. The version proposed in [24] therefore learns a dictionary of filters by optimizing the following objective function:

$$\min_{\mathbf{M},\{\mathbf{t}_i\}} \sum_i \|\mathbf{x}_i - \mathbf{M}\mathbf{t}_i\|_2^2 + \lambda_{\mathit{learn}} \|\mathbf{t}_i\|_1\,, \tag{2}$$

where the ℓ_1-norm enforces sparsity on the $\mathbf{t}_i$ vectors.

Eq. (2) looks for a dictionary $\mathbf{M}$ so that the images $\mathbf{x}_i$ can be reconstructed from only a few columns of $\mathbf{M}$ by comput-



(a) (b) (c)

Figure 2: Filter banks used in our experiments. (a) & (b) Filter banks learned using the *OLS* algorithm on the CIFAR-10 and Caltech datasets, respectively. (c) The handcrafted Leung-Malik filter bank taken from [20].

ing the product $\mathbf{M}\mathbf{t}_i$. The sparseness in the $\mathbf{t}_i$ vectors is enforced by the last term. λ_{learn} is a regularization parameter that establishes the relative importance of the reconstruction error $\|\mathbf{x}_i - \mathbf{M}\mathbf{t}_i\|_2^2$ with respect to the regularization term $\|\mathbf{t}_i\|_1$. Moreover, the dictionary is overcomplete: $\mathbf{M}$ has more columns than rows, and this gives us the degrees of freedom that we need in order to be able to choose among all the possible representations a sparse one.

Eq. (2) was introduced for small patches only, and using it on possibly large images is slow and difficult, as many coefficients in $\mathbf{M}$ would have to be optimized simultaneously. We therefore adopt here a convolutional approach, where the matrix-vector product is replaced by a convolution. This is possible if we assume that the local properties of images are translation invariant, which seems reasonable. As a side effect, we get a strongly overcomplete representation. [37] and [19] use a similar approach.

The optimization problem in Eq. (2) hence becomes:

$$\min_{\{\mathbf{f}^j\},\{\mathbf{t}_i^j\}} \sum_i \left(\left\| \mathbf{x}_i - \sum_j \mathbf{f}^j * \mathbf{t}_i^j \right\|_2^2 + \lambda_{learn} \sum_j \left\| \mathbf{t}_i^j \right\|_1 \right) , \quad (3)$$

where the $\mathbf{f}^j$s are linear filters and $*$ denotes the convolution operator. The $\mathbf{t}_i^j$s can now be seen as a set of images with the same size as the $\mathbf{x}_i$ images, whose cardinality is equal to that of the filter bank. Similar intermediate representations have been called "feature maps" in the Convolutional Neural Networks literature [17].

The original problem in Eq. (2) was optimized using stochastic gradient descent with clipping [5], and it is easy to also use stochastic gradient descent to find the coefficients of the $\mathbf{f}^j$ filters, alternatively optimizing the filters $\mathbf{f}^j$ and the $\mathbf{t}_i^j$s in Eq. (3).

We evaluated many choices for the regularization parameter λ_{learn}. While there is no guarantee that stochastic gradient descent provides the optimal solution, the optimization consistently converges from random initializations to extremely similar solutions for a large interval of λ_{learn} values. For example, in the case of CIFAR-10, we obtained a stable solution for $\lambda_{learn} \in [0.01, 3]$, though at different speeds. The learned filter banks are shown in Figs. 2(a) and 2(b).

The optimization algorithm, however, revealed to be very sensitive to the gradient descent steps choice both for filters and coefficients. For further investigation of the solution stability, we performed an experiment where Leung-Malik (*LM*) filters [20], depicted in Fig. 2(c), were used as initialization. In few iterations, the filter bank dramatically changes its aspect and converges to a solution similar to the one reported in Fig. 2(a).

3.1.2 Handcrafted filters

In addition to learned filters, we have also performed feature extraction with two handcrafted filter banks:

- The Leung-Malik (*LM*) filter bank [20]. It is composed of 2 Gaussian derivative filters at 6 orientations and 3 scales, 8 Laplacian of Gaussian filters and 4 Gaussian filters, for a total of 48 filters.
- A filter bank constituted by 49 randomly generated filters (*RND*), with the exception of the first one that is set to be uniform.

3.1.3 Pre-processing and whitening

We used grayscale images only, therefore the first preprocessing step transformed input color images into a grayscale representation in $[-1, 1]$. Since we had to deal with convolutions, we replicated the borders in order to exploit the full image information.

Removing the linear dependencies between the coefficients of the images, or whitening, revealed to be fundamental for the convergence of Eqs. (3) and (4). A whitening operation can be learned from the covariance matrix $\mathbf{C}$ of the original data [11]. By applying an eigenvalue decomposition to $\mathbf{C}$, $\mathbf{C} = \mathbf{E}\mathbf{D}\mathbf{E}^\top$, a whitening matrix $\mathbf{W}$ can be computed as $\mathbf{W} = \mathbf{E}\mathbf{D}^{-1/2}\mathbf{E}^\top$.

Similarly to Eq. (2), this is not really practical for large images. Nonetheless, owing to the shift invariance of image statistics, $\mathbf{W}$ describes a per pixel linear operation that is independent of translation, hence we were able to efficiently implement it as a convolution.

3.1.4 Using the filters for feature extraction

In our evaluations, we used these filters $\mathbf{f}^j$ to extract features $\mathbf{t}^j$ from an image $\mathbf{x}$ in three different ways:

- Sparse features with gradient descent (*SPARSEGD*). The $\mathbf{t}^j$s are obtained by minimizing the following objective function by gradient descent:

$$\min_{\{\mathbf{t}^j\}} \left\| \mathbf{x} - \sum_j \mathbf{f}^j * \mathbf{t}^j \right\|_2^2 + \lambda_{extract} \sum_j \left\| \mathbf{t}^j \right\|_1 . \quad (4)$$

This optimization is the same as the one posed in Eq. (3) after fixing the filters $\mathbf{f}^j$ and considering only the given image. Note that, in our experiments, the regularization parameter $\lambda_{extract}$ can be different from λ_{learn}.

- Sparse features with Matching Pursuit (*SPARSEMP*). The $\mathbf{t}^j$s are obtained using the Matching Pursuit algorithm [1]. Fixing the number of non-zero coefficients is equivalent to choosing a proper value for the $\lambda_{extract}$ parameter in Eq. (4). Nonetheless, Matching Pursuit is dramatically slower, which prevented us from using it to optimize Eq. (3).
- Features computed by direct convolution (*CONV*). The $\mathbf{t}^j$s are obtained by direct convolution, without any sparsity constraint:

$$\mathbf{t}^j = \mathbf{f}^j * \mathbf{x},\ \forall j\ . \tag{5}$$

This is much faster than the two previous options. It is not, however, unrelated to them, since it corresponds to the initialization step for the feature maps required by both algorithms.

3.1.5 Rectification

Before the pooling stage, we apply some non-linear operation to the feature maps $\mathbf{t}^j$, as it is usually done in multilayer architectures. This operation gives a new set of feature maps $\mathbf{u}^j$. Again, we tried different possibilities:

- Taking the absolute values of the coefficients of the $\mathbf{t}^j$ vectors (*ABS*). The m-th coefficient $\mathbf{u}^j[m]$ of the $\mathbf{u}^j$ vectors is simply taken to be: $\mathbf{u}^j[m] = \left|\mathbf{t}^j[m]\right|$. This operation was identified as very effective in [12] for recognition performance despite its simplicity.
- Separating the negative coefficients from the positive ones (*POSNEG*). If each submap is composed by N coefficients, the elements $\mathbf{u}^j[m]$ of the $\mathbf{u}^j$ vectors are taken to be:

$$\mathbf{u}^j[m] = [\mathbf{t}^j[m]]^+,\ \mathbf{u}^{N+j}[m] = [-\mathbf{t}^j[m]]^+\ , \tag{6}$$

where $[x]^+ = x$ if $x > 0$ and 0 otherwise. This doubles the number of coefficients in the $\mathbf{u}^j$ vectors.

3.2. Pooling stage

This stage pools the coefficients of the $\mathbf{u}^j$ vectors to provide invariance to small displacements and distortions. The choice of having a pooling stage is based on two relevant aspects:

- From a biological perspective, the pooling stage corresponds to the complex cells' layers in Hubel and Wiesel's model of V1 cortex [10]. The role that pooling holds is that of enabling a certain degree of invariance to minor pose and appearance changes. The importance of pooling layers is also acknowledged by their employment in Convolutional Neural Networks [17].
- From a computational perspective, plain descriptors have a dimensionality that is too high for practical applications. The presence of a downsampling step is therefore vital for subsequent operations.

We tried three different pooling mechanisms found in literature:

- Gaussian pooling (*GAUSS*). This was used in [31]: the $\mathbf{u}^j s$ are first convolved with a Gaussian filter, then downscaled by a factor that is a multiple of 2. We have empirically observed that a 5×5 filter with $\sigma = 2.0$ gave the best results in many cases, and we have therefore adopted these values [1].
- Average pooling (*BOXCAR*). This is similar to *GAUSS*, except that we use a boxcar filter.
- Maximum value pooling (*MAX*). We retain the maximum absolute value in a given neighborhood. This was used for example in [30, 12], and also evaluated in [3].

The feature maps for a given image after pooling will be denoted as $\mathbf{v}^j$ below.

3.3. Classification

The last step of our pipeline applies a classifier to the unitary normalized vectors obtained from the previous stages. Since the two datasets of choice have different cardinalities and are composed by images with different resolutions, we have adopted different strategies for the classification step.

3.3.1 CIFAR-10

The CIFAR-10 images have a resolution of 32×32 pixels. The feature maps after pooling $\mathbf{v}^j$ are nevertheless very large, and therefore a dimensionality reduction step before classification is necessary. We investigated the following methods:

- No dimensionality reduction (*NONE*).
- Principal Component Analysis (*PCA*).
- Local Discriminant Embedding (*LDE*) [9], with a power regularization fixing the signal to noise ratio to 15%.
- Random Projections (*RP*). We tried random projections because they can be applied to sparse signals with limited information loss. [6].

In both the PCA and the LDE case, a normalization to unit norm is performed after the projection, as it is deemed to give significant improvements on the final result [9]. In order to choose the best size of the eigenspace, for each specific configuration we performed an extensive cross-validation for all dimensions in a range $d = \{8, \ldots, 256\}$, and chose the value that scored best in a Nearest-Neighbor classification.

[1] We have performed an extensive evaluation of the different pooling parametrizations. Please refer to the supplementary material for the quantitative results of our investigations.

We then apply one of the two following classification methods on the feature maps after dimension reduction:

- Nearest Neighbor classification (*NN*). It provides a direct measure of the discriminative capabilities of the previous steps.
- Support Vector Machines (*SVM*). They are commonly adopted in pipelines similar to ours and usually achieved the best results [2].

We also tried other classifiers: Feed-Forward Neural Networks, ensembles of Classification Trees, and Naïve Bayes classifiers. As they did not give better results than *SVM*s, we do not report them here.

3.3.2 Caltech-101

The resolution of the Caltech-101 images is much larger than the one of the CIFAR-10 images, and the direct classification approach we used for CIFAR-10 is not possible anymore. Instead, and like [12] and [37], we use the Spatial Pyramid Matching (SPM) algorithm [16]. We use the code provided by [16] at the top of our architecture, and pass the resulting pyramidal histograms to an SVM [3].

We followed the same testing procedure reported in [37], denoted in the following as *SPM*. Input images are resized so that their shortest dimension is 150 pixels. 16×16 patches are extracted from the feature maps after pooling with a stride of 2 pixels and are used as input to the SPM algorithm. We build a dictionary containing 500 words by running the K-Means algorithm over the feature maps corresponding to 300 randomly chosen images, and use this dictionary to build a three-level pyramid.

As for the CIFAR-10 case, we tried both approximate Nearest Neighbor classification (*NN*) and Support Vector Machines (*SVM*).

4. Results and Discussion

4.1. Evaluation datasets

The CIFAR-10 dataset [32, 14] is a hand labelled subset of a larger dataset consisting of 32×32 images collected from the Web [32]. The images exhibit large variability in pose, appearance, scale, and background composition, and some are affected by severe distortions. These reasons justify the increasing popularity that it is gaining in the computer vision and machine learning community [27, 36, 15].

The above mentioned characteristics make the CIFAR-10 dataset suitable for our needs, since it avoids the common pitfalls involved with the uncontrolled exploitation of natural images [25]. Moreover, the low dimensionality of the images enables us to perform an extensive exploration of the parameter space, which would be prohibitively costly with other datasets.

We have also chosen to do additional experiments using the Caltech-101 dataset [7], which is widely acknowledged as a reference dataset in the computer vision community. Caltech-101 contains images from 101 different categories (with an additional background category). Since some categories have a relatively small number of samples, the most common training procedures use either 15 or 30 randomly chosen images per category, keeping the remaining ones for testing.

4.2. Protocol

Because of the large number of different combinations for our pipeline and the fact that most of them depend on parameters, we first performed thorough experiments on the CIFAR-10 images converted to grayscale and downsampled to 16×16 pixels, as computational costs for extensive experiments on full resolution images or Caltech-101 are much higher. This allowed us to identify trade-offs and the best components of our architecture. Once the most effective combinations were determined, we validated them on the original 32×32 images of the CIFAR-10 dataset and, after having identified the proper parametrization of each component, on Caltech-101 images. In the case of Caltech-101, we had to restrict the number of experiments due to computation times and adopt *ABS* as a non-linearity to avoid the doubling of descriptor's size. The results, however, are consistent with those obtained on CIFAR-10 and validate our interpretation.

We report here only the small subset of our trials that illustrates our main findings. Extensive additional results are provided as supplementary material.

4.3. Experimental Results and Discussion

4.3.1 Sparsity is not necessarily required for classification

Our first experiment aimed to evaluate the influence of the way the features are extracted on the recognition rate. Figure 3 reports the results of our classification pipeline for different filter banks and different feature extraction methods. The other components were set to *POSNEG*, *GAUSS*, *PCA*, *SVM*, which is one of the best combinations we found.

A general rule is that, as sparsity increases when computing the features, the recognition rate drops dramatically. Moreover, the value of $\lambda_{extract}$ used for *SPARSEGD* must be much smaller than the one of λ_{learn} used to learn the filter bank. This was already observed in [28], which noted that a strong sparsity is important in learning the feature extractors, but harmful during classification.

But more importantly, using simple convolutions (*CONV*) performs systematically at least as good as

[2] Experiments were performed using the LIBSVM library (http://www.csie.ntu.edu.tw/~cjlin/libsvm).

[3] We used the BSVM implementation bundled within the libHIK library (http://www.cc.gatech.edu/cpl/projects/libHIK).

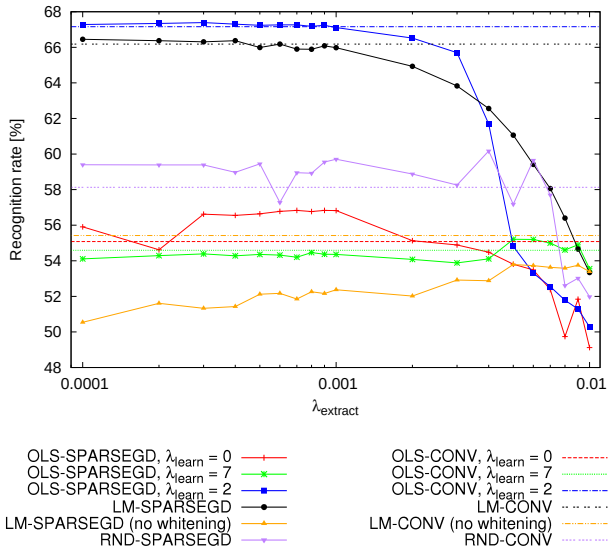


Figure 3: Recognition rate on the CIFAR-10 dataset as a function of the representation's sparsity and of the chosen filter bank.

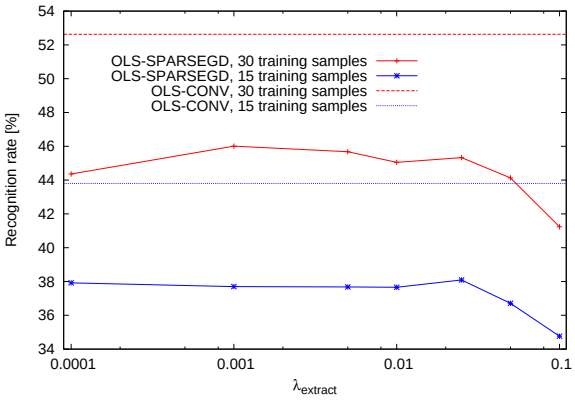


Figure 4: Recognition rate on the Caltech-101 dataset as a function of the representation's sparsity.

enforcing sparsity (*SPARSEGD*). The $\mathbf{t}_i$s are significantly sparser when using *SPARSEGD* instead of *CONV*, but this does not have an influence on the recognition rate. This is true whatever the way the filter bank was computed, for both the CIFAR-10 and Caltech-101 benchmarks (see Fig. 4). Enforcing sparsity clearly does not help here.

To investigate more when sparsity can be useful, we ran the same experiments on images from the datasets after corruption by noise. The most significant results are reported in Table 1. We have experimented with both Gaussian and structured noise, where the latter consists of randomly generated lines superimposed to the images. In all these experiments, we worked with the original 32×32 images of CIFAR-10. *SPARSEGD* performs well in presence of strong Gaussian noise, but does not help for structured noise, as

Table 1: Recognition rates in presence of noise for different feature extraction methods using learned filters and an SVM as a classifier. Image intensities are normalized in $[0, 1]$. For the structure noise experiments, random lines were superimposed to the images at random positions. SPARSEGD is more interesting than CONV only in case of strong Gaussian noise, for some values of $\lambda_{extract}$.

Method	$\lambda_{extract}$	$\|\mathbf{t}\|_0$	**Rec. Rate [%]**
low Gaussian noise ($\sigma = 0.01$)			
CONV		**1.00**	**69.44**
SPARSEGD	0.0001	0.83	68.66
SPARSEGD	0.0005	0.58	67.07
SPARSEGD	0.001	0.43	64.54
SPARSEGD	0.005	0.11	54.37
strong Gaussian noise ($\sigma = 0.14$)			
CONV		1.00	60.30
SPARSEGD	0.0001	0.88	61.89
SPARSEGD	**0.0005**	**0.69**	**63.54**
SPARSEGD	0.001	0.55	63.28
SPARSEGD	0.005	0.17	59.94
low structured noise (1 random line)			
CONV		**1.00**	**48.53**
SPARSEGD	0.0005	0.51	47.00
SPARSEGD	0.005	0.09	31.75
strong structured noise (1 to 3 random lines)			
CONV		**1.00**	**35.20**
SPARSEGD	0.0005	0.49	33.51
SPARSEGD	0.005	0.09	15.08

it focuses its efforts around the noisy area, skipping the parts of the images that convey meaningful information. Nonetheless, since the original images of the datasets are mostly free of noise, this is a property unexploited when evaluating algorithms on these benchmarks.

Another relevant result is the impact of the sparsifying algorithm, either Gradient Descent (*SPARSEGD*) or Matching Pursuit (*SPARSEMP*), on the final recognition rate as reported in Fig. 5. Figure 6 also provides visual results. It is not straightforward to compare the two methods as they depend on different parameters. We chose to plot the recognition rate as a function of the ℓ_0-norm of the $\mathbf{t}$ vectors, as the two algorithms provide the lowest reconstruction error they can reach for a given value of this norm. For a given ℓ_0-norm of the $\mathbf{t}$ vectors, the performance are significantly worse for *SPARSEMP* than *SPARSEGD* while the reconstruction errors are similar. A possible explanation for the very bad performance of the Matching Pursuit algorithm is that, because it works locally, it tends to focus on details that are specific to the given instance, hence it tends to in-

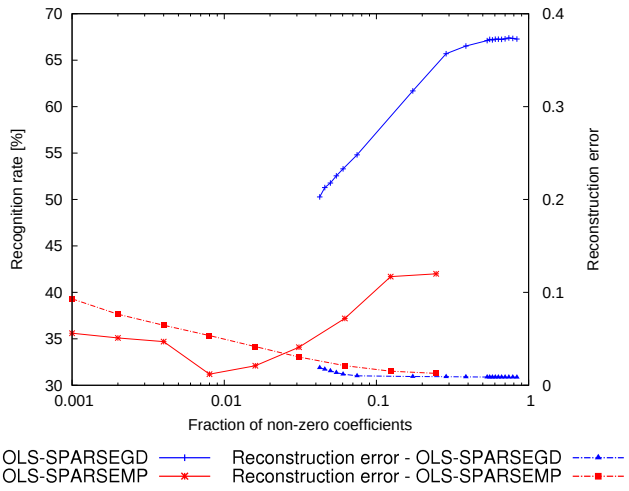


Figure 5: Recognition rate comparison between *SPARSEMP* and *SPARSEGD* when different degrees of representation's sparsity are requested (left axis). The reconstruction error (right axis) is computed as the ℓ_2 norm of the squared differences between the original image and the reconstruction (both normalized in $[0, 1]$).

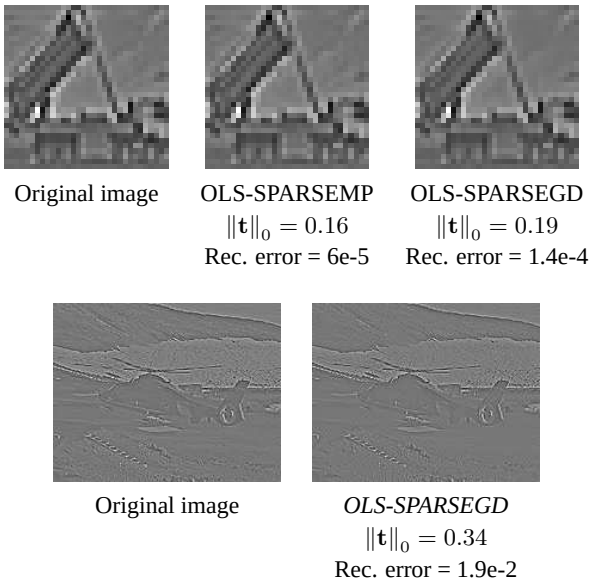


Figure 6: **First row**. An image from the CIFAR-10 dataset whitened, and its reconstructions from the $\mathbf{t}$ vectors obtained with different algorithms and filter banks. The given reconstruction error is the first term of Eq. (4). While similar in terms of sparsity of the features, of reconstruction error, and visual quality of the reconstruction, Matching Pursuit performs significantly worse than Gradient Descent. **Second row**. An image taken from the Caltech-101 dataset and whitened, along with the image reconstructed from the $\mathbf{t}$ vectors using *OLS-SPARSEGD*. Matching Pursuit is too slow to be evaluated on large images such as those present in the Caltech-101 dataset.

crease intra-class dissimilarities.

A last remark on this aspect is that, in all the experiments we have carried out and reported in the supplemental material, irrespectively of the chosen feature extraction and pooling strategies, the results after pooling are dense. In architectures that employ pooling stages, sparsity is therefore a temporary condition only.

4.3.2 Sparsity is important when learning the filters

Although a sparse representation is not necessarily important for feature extraction during classification, and can even hurt the recognition rate when an unsuited sparsification algorithm is used, we found that it is still important for *learning* the filters.

Figure 3 clearly depicts the advantage gained by adopting a filter bank learned with sparsity constraints, compared to relying on handcrafted or random filters. This is true even when using simple convolutions (*CONV*) to extract features. While the handcrafted LM filters perform almost as well as the learned ones, this is true only when they are whitened, since the performance of the LM filter bank when the whitening step is removed is even worse than the performance of whitened random filters.

4.3.3 Best results

Thanks to our extensive experiments, we could identify the components and their parameters that perform best. Due to lack of space, the complete evaluation results are not shown in this paper but in the supplemental material. The best configuration extracts features by convolution with learned filters (*OLS-CONV*). *POSNEG* is the best non-linear operation. Pooling features is indispensable, and pooling by 4 times downscaling after having smoothed feature maps with a 5×5 Gaussian filter with $\sigma = 2$ (*GAUSS*) performs best. *LDE* and *PCA* for subspace projection give almost identical results. *SVM*s perform definitively better than *NN* for final classification.

Despite its simplicity, our best architecture performs extremely well. The two-layer convolutional Deep Belief Network presented in [15] achieves a 78.9% recognition rate by using color images and an unsupervised pre-training set of 1.6 million images, while our architecture accomplishes a 75.18% (averaged on 5 random dataset splits, standard deviation $= 0.27$) by using grayscale images only. Moreover, our model outperforms specifically designed, complicated, color-based machine learning architectures like the factorized third-order Boltzmann Machine proposed in [27] or the improved version of the 2009 PASCAL image classification

challenge winning system presented in [36].

5. Conclusions

We have performed an in-depth analysis of sparse representations in image classification. Our experimental results suggest that solely enforcing sparsity is not helpful in terms of recognition rate, at least when the level of noise remains reasonable. We found that while sparsity was not helpful during classification, with plain convolution with the filters giving equal results in terms of recognition rate, it is important when learning the feature dictionary itself. Given the high computational burden involved in sparse coding and the increasing interest in biologically inspired multilayer architectures, this insight heavily impacts on the design strategies for image descriptors.

References

[1] F. Bergeaud and S. Mallat. Matching Pursuit of Images. In *ICIP*, 1995.

[2] Y.-L. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning Mid-Level Features for Recognition. In *CVPR*, 2010.

[3] Y.-L. Boureau, J. Ponce, and Y. LeCun. A Theoretical Analysis of Feature Pooling in Visual Recognition. In *ICML*, 2010.

[4] M. Brown, G. Hua, and S. Winder. Discriminative Learning of Local Image Descriptors. *PAMI*, 2010.

[5] I. Daubechies, M. Defrise, and C. D. Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *CPAM*, 2004.

[6] D. L. Donoho. Compressed Sensing. *TIT*, 2006.

[7] L. Fei-Fei, R. Fergus, and P. Perona. Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. In *CVPR*, 2004.

[8] G. E. Hinton. Learning to Represent Visual Input. *RSTB*, 2010.

[9] G. Hua, M. Brown, and S. Winder. Discriminant Embedding for Local Image Descriptors. In *ICCV*, 2007.

[10] D. H. Hubel and T. N. Wiesel. Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex. *JPHYSIO*, 1962.

[11] A. Hyvärinen, J. Hurri, and P. O. Hoyer. *Natural Image Statistics*. Springer-Verlag, 2009.

[12] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun. What Is the Best Multi-Stage Architecture for Object Recognition? In *ICCV*, 2009.

[13] K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun. Fast Inference in Sparse Coding Algorithms With Applications to Object Recognition. Technical report, NYU, 2008.

[14] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, 2009.

[15] A. Krizhevsky. Convolutional Deep Belief Networks on CIFAR-10. Technical report, UOFT, 2010.

[16] S. Lazebnik, C. Schmid, and J. Ponce. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In *CVPR*, 2006.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *PIEEE*, 1998.

[18] H. Lee, C. Ekanadham, and A. Y. Ng. Sparse Deep Belief Net Model for Visual Area V2. In *NIPS*, 2007.

[19] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. In *ICML*, 2009.

[20] T. Leung and J. Malik. Representing and Recognizing the Visual Appearance of Materials Using Three-Dimensional Textons. *IJCV*, 2001.

[21] D. G. Lowe. Distinctive Image Features from Scale-Invariants Keypoints. *IJCV*, 2004.

[22] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Discriminative Learned Dictionaries for Local Image Analysis. In *CVPR*, 2008.

[23] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-Local Sparse Models for Image Restoration. In *ICCV*, 2009.

[24] B. A. Olshausen and D. J. Field. Sparse Coding With an Overcomplete Basis Set: A Strategy Employed by V1? *VISR*, 1997.

[25] N. Pinto, D. D. Cox, and J. J. DiCarlo. Why Is Real-World Visual Object Recognition Hard? *PLoS*, 2008.

[26] M. A. Ranzato, Y.-L. Boureau, and Y. LeCun. Sparse Feature Learning for Deep Belief Networks. In *NIPS*, 2007.

[27] M. A. Ranzato and G. E. Hinton. Modeling Pixel Means and Covariances Using Factorized Third-Order Boltzmann Machines. In *CVPR*, 2010.

[28] M. A. Ranzato, F.-J. Huang, Y. Boureau, and Y. LeCun. Unsupervised Learning of Invariant Feature Hierarchies With Applications to Object Recognition. In *CVPR*, 2007.

[29] M. A. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient Learning of Sparse Representations With an Energy-Based Model. In *NIPS*, 2006.

[30] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust Object Recognition With Cortex-Like Mechanisms. *PAMI*, 2007.

[31] E. Tola, V. Lepetit, and P. Fua. DAISY: An Efficient Dense Descriptor Applied to Wide-Baseline Stereo. *PAMI*, 2010.

[32] A. Torralba, R. Fergus, and W. T. Freeman. 80 Million Tiny Images: A Large Dataset for Non-Parametric Object and Scene Recognition. *PAMI*, 2008.

[33] W. E. Vinje and J. L. Gallant. Sparse Coding and Decorrelation in Primary Visual Cortex During Natural Vision. *SCIENCE*, 2000.

[34] J. Wright, Y. Ma, J. Mairal, G. Sapiro, T. S. Huang, and S. Yan. Sparse Representation for Computer Vision and Pattern Recognition. *PIEEE*, 2010.

[35] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification. In *CVPR*, 2009.

[36] K. Yu and T. Zhang. Improved Local Coordinate Coding Using Local Tangents. In *ICML*, 2010.

[37] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional Networks. In *CVPR*, 2010.